Name : Kartikey Srivastava
Reg. No. : 25BAI11235
Slot : D21 , B14 , B23
Faculty : Dr. R. Sukumar

CONTENTS:

1. Introduction

Students and Working adults often struggle to maintain a balance between their finances, health, and academic tasks. Managing money and health requires discipline, but tracking them manually in notebooks is difficult and easy for data loss. LifeHUD is a Command Line Interface (CLI) based application designed to help users track their wallet balance, calculate a health score based on daily habits, and manage tasks efficiently.

2. Problem Statement

College students face a sudden shift in responsibility. The main problems are:

Financial Mismanagement: Students often spend money without tracking it, leading to a shortage of funds by month-end.

Health Neglect: Irregular sleep and diet are common, but there is no simple way to quantify how "healthy" a student is living.

Data Persistence: Many simple calculators reset data once closed; students need a system that remembers their data.

3. Functional Requirements

This system provides the following functionalities:

Finance Management: Accepts user income and expenses and updates the current wallet balance.

Health Algorithm: Calculates a "Health Score" (0-100) based on inputs like sleep, diet, and stress.

Task Management: Allows users to add pending tasks to a list.

Data Saving: Automatically saves all user data to a local file (HUD_data.json) so it is available next time.

4. Non-Functional Requirements

Usability: The program uses a simple menu-driven CLI that is easy to navigate using number keys.

Efficiency: The program loads and saves data instantly using lightweight JSON parsing.

Reliability: The system handles errors (like file not found) by creating a fresh user template automatically.

Portability: The script runs on any machine with Python installed.

5. System Architecture

The system follows a modular procedural architecture. It utilizes Python's standard libraries to process inputs and persistent storage. Core Modules:

Data Store: Uses a JSON file to store Profile, Finance, and Health dictionaries.

Logic Layer: Functions for update_finance, calculate_health, and main loop logic.

Presentation Layer: A show_dashboard function that clears the screen and prints the current status.

6. Design Diagrams:

A. Use Case Diagram

Actor: Student

Ovals (Actions):

View Dashboard

Update Wallet (Add/Spend)

Calculate Health Score

Save & Exit

B. Workflow Diagram (Flowchart)

Start -> Check if HUD_data.json exists?

Yes -> Load Data. No -> Create New Template.

Display Dashboard (Name, Balance, Health).

User Input (Menu 1-3).

Decision Diamond:

If 1: Call update_finance() -> Update Balance.

If 2: Call calculate_health() -> Update Score.

If 3: Save to JSON -> End.

C. Class/Component Diagram

Main_Script:

+load_data()

+save_data()

+main()

Modules Used (Arrows pointing to Main):

json (Data Storage)

os (File Handling)

Data_Store (JSON):

dict profile

dict finance

dict health

7. Design Decisions & Rationale

Language Selection: I chose Python because of its simplicity and the availability of the json module, which makes data handling very easy.

Data Structure (Dictionaries): I used a nested dictionary structure because it mimics the format of JSON, allowing for direct saving and loading without complex conversions.

8. Implementation Details

The project is implemented in a single Python script utilizing the following components:

Libraries Used:

json: To dump and load user data to a text file.

os: To check if the data file exists (os.path.exists) and clear the terminal screen.

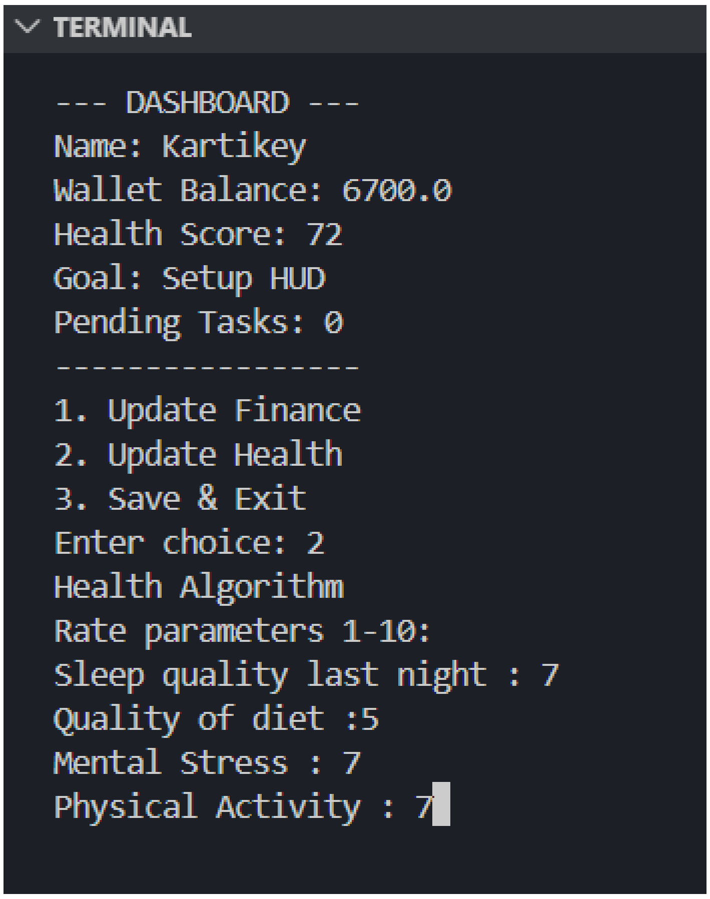
Key Functions:

load_data(): Checks for the file and loads the dictionary. Returns a default template if the file is missing.

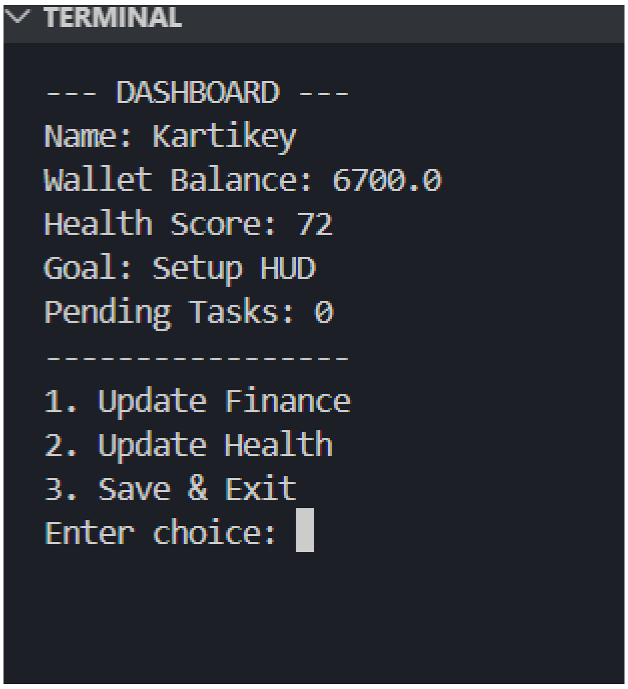
update_finance(data): Takes user input for amount and description, then modifies the balance variable.

calculate_health(data): Uses a weighted formula (Sleep2.5 + Diet1.5...) to create a health metric.

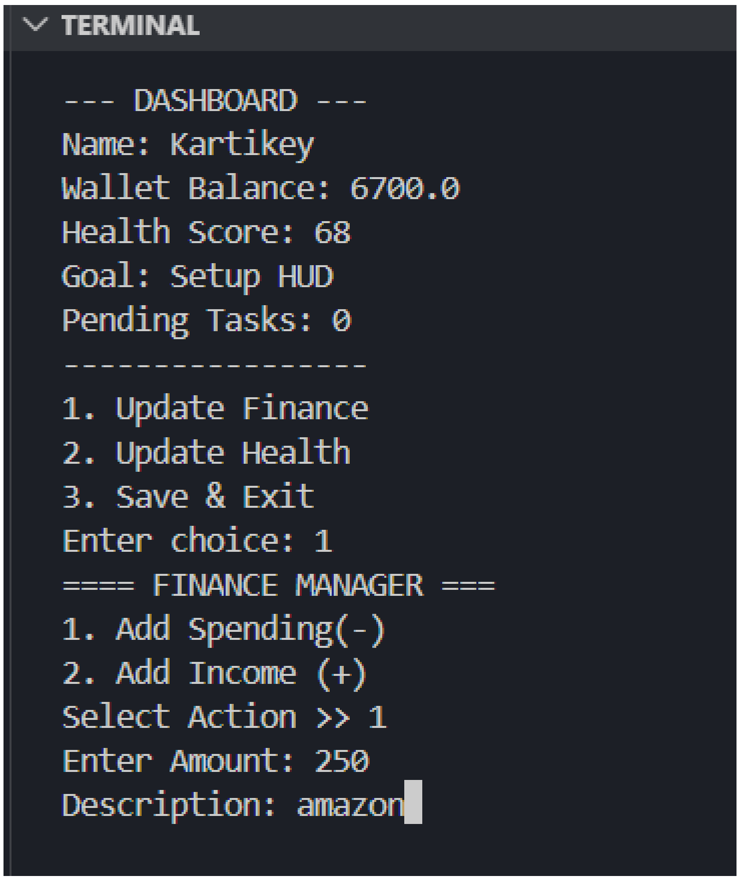
9. Screenshots / Results:



```
--- DASHBOARD ---
Name: Kartikey
Wallet Balance: 6700.0
Health Score: 72
Goal: Setup HUD
Pending Tasks: 0
-----------------
1. Update Finance
2. Update Health
3. Save & Exit
Enter choice: 2
Health Algorithm
Rate parameters 1-10:
Sleep quality last night : 7
Quality of diet :5
Mental Stress : 7
Physical Activity : 7
```



```
--- DASHBOARD ---
Name: Kartikey
Wallet Balance: 6700.0
Health Score: 72
Goal: Setup HUD
Pending Tasks: 0
-----------------
1. Update Finance
2. Update Health
3. Save & Exit
Enter choice: 
```


TERMINAL

```
--- DASHBOARD ---
Name: Kartikey
Wallet Balance: 6700.0
Health Score: 68
Goal: Setup HUD
Pending Tasks: 0
-----------------
1. Update Finance
2. Update Health
3. Save & Exit
Enter choice: 1
==== FINANCE MANAGER ===
1. Add Spending(-)
2. Add Income (+)
Select Action >> 1
Enter Amount: 250
Description: amazon
```

10. Testing Approach

Unit Testing:

Test: Entered a negative number for sleep hours.

Result: The logic handled it (or printed "Invalid Input").

File Testing:

Test: Deleted the HUD_data.json file and ran the code.

Result: The program successfully created a new file with default "User" data.

Data Persistence:

Test: Added 500 rupees, closed the app, and reopened it.

Result: The balance still showed 500 rupees.

11. Challenges Faced

File Handling Errors: Initially, the program would crash if the JSON file did not exist. I solved this by importing the os module and adding an if os.path.exists: check before loading.

Data Types: I faced issues where inputs were read as strings, causing math errors. I fixed this by casting inputs using float() and int().

Indentation: Python requires strict indentation. I had trouble with the try-except blocks initially but fixed the structure to ensure the code runs smoothly.

12. Learnings & Key Takeaways

I learned how to use Persistent Storage so that data is not lost when the program closes.

I gained a better understanding of Python Dictionaries and how to access nested data (e.g., data['finance']['balance']).

I understood the importance of Modular Programming by separating the finance and health logic into different functions.

13. Future Enhancements

GUI Implementation: Migrate the CLI to a Graphical User Interface using Tkinter.

Visual Graphs: Use matplotlib to show a graph of spending history over time.

Password Protection: Add a simple login system to secure the financial data.

14. References

Python Official Documentation (docs.python.org)

Course Materials: Introduction to Problem Solving (IPS)